Formateo de moneda e internacionalización con Python

Yan Orestes
17/02/2020

¿Cómo formatear una moneda usando Python?

Soy programador de una tienda en línea brasileña y, recientemente, decidimos implementar la venta de productos a Portugal, con el fin de incrementar nuestra clientela. Para hacer esto, necesitamos convertir los precios de Real a Euro.

Tenemos una función ya implementada para realizar la conversión, la cual toma como parámetro el valor en Real como flotante y devuelve el valor en Euro, también flotante:

```
def convierte_real_a_euro(valor_en_real):
    ## implementação da conversão
    return valor_en_euro

valor_inicial_en_real = 50.0
valor_convertido_a_euro = convierte_real_a_euro(valor_inicial_en_real)

print(valor_convertido_a_euro)
```

Probemos nuestro código:

```
12.4593681
```

¡La función funciona bien! ¿Pero es este resultado suficiente? Después de todo, ¿qué significa todo este número suelto?

Tenga en cuenta que tenemos un valor con 7 cifras decimales (lo que dificulta la legibilidad), sin indicación de a qué nos referimos, de su significado. ¿Cuál es el significado de este número?

Necesitamos hacer que el valor convertido sea más legible, formatearlo. Queremos convertir ese número entero en **12,46 €**, el estándar de Portugal. ¿Cómo podemos hacerlo?

Redondeando un número con la función `round()`

El primer paso para formatear el número es redondearlo, después de todo, no podemos dejar un valor con 7 decimales a nuestros clientes.

Una solución simple a esto se logra a través de la función nativa de Python **round()**, que toma como parámetro el número a redondear y la cantidad de cifras decimales que queremos:

```
valor_redondeado_a_euro = round(valor_convertido_a_euro, 2)
print(valor_redondeado_en_euro)
```

Y ya tenemos el valor redondeado a 2 cifras decimales:

```
12.46
```

Reemplazando un carácter de una string con el método `replace()`

Ahora necesitamos transformar el "." en "," , para seguir el estándar. Para esto, podemos usar el método replace() de objetos string, especificando **qué** y **para qué** queremos reemplazar en los parámetros, así:

```
valor_en_euro_texto = valor_redondeado_en_euro.replace('.', ',')
print(valor_en_euro_texto)
```

Probemos el `replace()`:

```
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
AttributeError: 'float' object has no attribute
'replace' AttributeError: 'flotador' objeto no tiene atributo
'reemplazar'
```

¡Ups! ¡Recibimos una excepción!

La excepción, de tipo `AttributeError`, indica que el tipo flotante no tiene atributos ni métodos replace. Eso ocurre, pues el `replace()` es un método de la clase string, es decir, tenemos que transformar nuestro valor convertido de float en string. Vamos otra vez:

```
valor_en_euro_texto = str(valor_redondeado_en_euro).replace( '.' ,  ',' )
print(valor_en_euro_texto)
```

Y el resultado:

```
12,46
```

¡Ahora sí! De todos modos, solo necesitamos agregar el símbolo del euro, que se puede hacer con un simple formateo de string:

```
valor_en_euro_formateado = '{ €'.format(valor_en_euro_texto)
print(valor_en_euro_formateado)
```

Veamos cómo se ve nuestro valor ahora:

```
12,46 €
```

¡Excelente! Realmente logramos formatear nuestro valor en el estándar castellano de moneda.

Soporta más de una moneda y siguiendo diferentes estándares

Si tuviéramos un número mayor, como **1000.50**, y quisiéramos agrupar el número por miles, transformándolo en **1.000,50 €**, ¿Qué haríamos?

Además de un cuidado especial para no reemplazar el " . " agrupando por una " , " por el `replace()`, necesitamos un código que sepa dónde se debe realizar la agrupación.

Otro problema es que estamos restringidos al estándar castellano. ¿Y si, por ejemplo, quisiéramos abrir las puertas a los clientes estadounidenses?

El estándar ya no sería **12,46 €**, pero si **$ 12.46**. Otro caso sería devolver el valor en Real, por lo que el valor predeterminado sería **R$ 12,46**...

¿Necesitamos implementar una forma diferente de formatear la moneda para cada país?

Conociendo el módulo `locale`

Python nos ofrece un módulo completo dedicado a los servicios de internacionalización, lo que nos permite a los programadores abordar fácilmente posibles problemas culturales en una aplicación: el módulo **locale**.

Desde este módulo tenemos varias funciones que pueden ayudarnos a convertir el formato de ciertas especificidades, como el tiempo y el dinero.

En nuestro caso, podemos usar la función locale.currency(), pasando nuestro valor que queremos formatear como parámetro:

```
import locale

valor_formateado = locale.currency(12.4593681)
print(valor_formateado)
```

Vamos a probar:

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib64/python3.6/locale.py", line 262, in currency
    raise ValueError("Currency formatting is not possible using "
ValueError: Currency formatting is not possible using the 'C' locale.
```

¡Oye! Una vez más una excepción, esta vez del tipo `ValueError`, lo que indica que el formato de moneda no es posible mediante el `locale` predeterminado, que es el `C`.

El `C` es el `locale` más sencillo, sin las definiciones específicas para realizar la operación que queremos.

Definiendo un *locale* específico con la función `setlocale()`

Entonces necesitamos definir un `locale` en nuestro programa, para repasar el estándar. Para ello usaremos la función `locale.setlocale()`, que necesita dos argumentos:

`category`: La categoría que define el tipo de operación que se cambiará al locale especificado

`locale`: Una cadena que representa el *locale* que queremosEchemos un vistazo a las posibilidades de utilizar estos dos parámetros. Primero, las categorías permitidas son:

`locale.LC_CTYPE`- Funciones relacionadas con los caracteres

`locale.LC_COLLATE`- Orden de strings

`locale.LC_TIME`- Formateo de hora

`locale.LC_MONETARY`- Formateo de valores monetarios

`locale.LC_MESSAGES`- Visualización de mensajes

`locale.LC_NUMERIC`- Formateo de números

`locale.LC_ALL`- Combinación de todas las categorías

Como estamos tratando de formateo de monedas, usaremos la categoría `LC_MONETARY`.

En el caso del parámetro `locale`, las strings admitidas dependen del sistema en el que se ejecuta el código. Por ejemplo, en sistemas basados en UNIX, podemos ver una lista de *locales* registrados en nuestra máquina con el siguiente comando en la Terminal:

```
locale -a
```

En Windows, se vuelve un poco complicado. Incluso podemos ver una **lista de locales publicada por Microsoft**, pero posiblemente hay elementos en esa lista que no funcionan en su sistema y aún *locales* en su sistema que no están en la lista.Como estoy trabajando en Linux y quiero convertir a dólares estadounidenses, usaré el *locale* **en_US.UTF-8**:

```
import locale
```

```
locale.setlocale(locale.LC_MONETARY, 'en_US.UTF-8')
```

El método `setlocale()`devuelve una string que nos muestra el *locale* que elegimos:

```
en_US.UTF-8
```

Formateo de moneda con la función `currency()`

Ahora, solo usa la función `currency()`:

```
valor_en_dolar_formateado = locale.currency(12.4593681)
print(valor_en_dolar_formateado)
```

Y tenemos a cambio:

```
$12.46
```

¡Todo resuelto! Tenga en cuenta que la función `currency()` incluso nos redondeó el valor a nosotros, ¡eliminando nuestra necesidad de usar la función `round()`! Probemos con un valor más alto, para ver cómo se ve:

```
valor_en_dolar_formateado = locale.currency(15000.0)
print(valor_en_dolar_formateado)
```

El resultado:

```
$15000.00
```

De acuerdo, pero queríamos que los miles estuvieran agrupados en el formateo, ¿recuerdas? Para ello, además de enviar el valor, también podemos pasar otro argumento al método currency: el `grouping`.

El parámetro `grouping` espera recibir un boolean, que por estándar es **False**. Para agrupar, entonces, simplemente indique cómo **True**:

```
valor_en_dolar_formateado = locale.currency(15000.0, grouping=True)
print(valor_en_dolar_formateado)
```

Esta vez:

```
$15,000.00
```

¡Exactamente como el estándar estadunidense!

*Otros parámetros que podemos definir son **symbol**, que define si queremos el símbolo de moneda (como el **$**) y es **True** por estándar, e **international**, que define si queremos utilizar el símbolo de moneda internacional (como por ejemplo, **USD**) y es por estándar **False**.*

Conclusión

En esta publicación, comenzamos con la necesidad de formatear una moneda euro utilizando los estándares correctos.

Probamos, primero, funciones como `round()`, para redondear el valor y métodos como `replace()` y `format()` para ajustar la string de valor de Euro formateado, y confirmamos que es posible hasta cierto punto; funciona, pero podemos encontrar dificultades.

Por fin, conocemos el módulo `locale`, que consigue solucionar problemas de internacionalización sin exigir demasiado al programador. En nuestro caso, usamos la función `currency()` definido en ese módulo.

Para saber más

¿Quiere aprender más sobre el poder del módulo `locale` en Python? ¡Siempre es una buena idea leer la **documentación** para saber lo que podemos hacer!

¿Y? Ahora es más sencillo formatear una moneda, ¿no? ¡Y todavía sabemos dónde buscar si necesitamos un código que tenga en cuenta aspectos culturales! ¿Quieres aprender más sobre Python? Echa un vistazo a **nuestros cursos en Alura** ¡y sigue estudiando!

Puedes leer también:

- Comprensión de listas en Python
- Python datetime: trabajando con fechas

- Trabajando con precisión en números decimales en Python

ARTÍCULOS DE TECNOLOGÍA > PROGRAMACIÓN

CURSOS

Cursos de Programación
Lógica de Programación | Java

Cursos de Front End
HTML y CSS | JavaScript | React

Cursos de Data Science
Data Science | Machine Learning | Excel | Base de Datos | Data Visualization | Estadística

Cursos de DevOps
Docker | Linux

Cursos de Innovación y Gestión
Productividad y Calidad de Vida | Transformación Ágil | Marketing Analytics | Liderazgo y Gestión de Equipos | Startups y Emprendimiento